Correctness of Algorithms

Stefan D. Bruda

CS 317, Fall 2025

CORRECTNESS OF ITERATIVE ALGORITHMS

- For this course we establish correctness semi-formally
  - Rigorous correctness argument, but not necessarily formulated in a formal logic framework

- For this course we establish correctness semi-formally
  - Rigorous correctness argument, but not necessarily formulated in a formal logic framework
- Establishing the correctness of sequences of statements is generally easy
  - A simple argument that walks through the code usually suffices

Correctness of Iterative Algorithms

- For this course we establish correctness semi-formally
  - Rigorous correctness argument, but not necessarily formulated in a formal logic framework
- Establishing the correctness of sequences of statements is generally easy
  - A simple argument that walks through the code usually suffices
- Establishing the correctness of loops is best done by coming up with a loop invariant
  - Can choose some place in the loop (usually either the beginning or the end of the loop code) where the invariant is always true
  - The invariant must imply the desired property of the output (at the end of the loop)
  - That the invariant is indeed an invariant can be proven by induction over the number of the current iteration
    - Prove that the invariant is true at the start of the loop (Iteration 0)
    - Assume that the invariant is true at iteration k and then prove that it is also true at iteration $k + 1$
    - Make sure that the invariant establishes the desired correctness at the end of the loop

algorithm BINSEARCH(x, S, l, h):
```
    // S_{l...h} is a sorted sequence
    i ← l
    j ← h
    while i ≤ j do
        m ← (i + j)/2
        if S_m = x then return m
        else if S_m > x then j ← m − 1
        else i ← m + 1
    return −1
```

- Need to show that for return r:
  $S_r = x \vee r = -1 \wedge x \notin S_{l\ldots h}$

```
algorithm BinSearch(x, S, l, h):
    // S_{l...h} is a sorted sequence
    i ← l
    j ← h
    while i ≤ j do
        m ← (i + j)/2
        if S_m = x then return m
        else if S_m > x then j ← m − 1
        else i ← m + 1
    return −1
```

- Need to show that for return r: $S_r = x \vee r = -1 \wedge x \notin S_{l\ldots h}$
- Loop invariant, true at the beginning of every iteration:

algorithm BINSEARCH(x, S, l, h)**:**

```
// S_{l...h} is a sorted sequence
i ← l
j ← h
while i ≤ j do
    m ← (i + j)/2
    if S_m = x then return m
    else if S_m > x then j ← m − 1
    else i ← m + 1
return −1
```

- Need to show that for return r: $S_r = x \vee r = -1 \wedge x \notin S_{l \ldots h}$

- Loop invariant, true at the beginning of every iteration:

$$S_{(i+j)/2} = x \vee x \notin S_{l \ldots i-1} \wedge x \notin S_{j+1 \ldots h}$$

  - Clearly $x \notin S_{l \ldots i-1} \wedge x \notin S_{j+1 \ldots h}$ holds for $i = l$ and $j = h$ so the invariant is true at the start of the loop
  - If $S_{(i+j)/2} = x$ then the loop terminates (there is no next iteration)
  - Otherwise ($x \notin S_{l \ldots i-1} \wedge x \notin S_{j+1 \ldots h}$ is true):
    - If $S_{m=(i+j)/2} > x$ then $S_{m \ldots j} \geq S_m > x$ and so $x \notin S_{m \ldots h} \wedge x \notin S_{l \ldots i-1}$ This shows that the invariant is true at the next iteration since $j \leftarrow m - 1$
    - If $S_{m=(i+j)/2} < x$ then $S_{l \ldots i} < S_m < x$ and so $x \notin S_{i \ldots m} \wedge x \notin S_{j+1 \ldots h}$ This shows that the invariant is true at the next iteration since $j \leftarrow m + 1$

INVARIANT EXAMPLE

```
algorithm BinSearch(x, S, l, h):
    // S_{l...h} is a sorted sequence
    i ← l
    j ← h
    while i ≤ j do
        m ← (i + j)/2
        if S_m = x then return m
        else if S_m > x then j ← m − 1
        else i ← m + 1
    return −1
```

- Need to show that for return r:
  $S_r = x \vee r = -1 \wedge x \notin S_{l\dots h}$

- Loop invariant, true at the beginning of every iteration:

$$S_{(i+j)/2} = x \vee x \notin S_{l\dots i-1} \wedge x \notin S_{j+1\dots h}$$

  - Clearly $x \notin S_{l\dots i-1} \wedge x \notin S_{j+1\dots h}$ holds for $i = l$ and $j = h$ so the invariant is true at the start of the loop
  - If $S_{(i+j)/2} = x$ then the loop terminates (there is no next iteration)
  - Otherwise ($x \notin S_{l\dots i-1} \wedge x \notin S_{j+1\dots h}$ is true):
    - If $S_{m=(i+j)/2} > x$ then $S_{m\dots j} \geq S_m > x$ and so $x \notin S_{m\dots h} \wedge x \notin S_{l\dots i-1}$
      This shows that the invariant is true at the next iteration since $j \leftarrow m - 1$
    - If $S_{m=(i+j)/2} < x$ then $S_{l\dots i} < S_m < x$ and so $x \notin S_{i\dots m} \wedge x \notin S_{j+1\dots h}$
      This shows that the invariant is true at the next iteration since $j \leftarrow m + 1$
- How the invariant establishes correctness when the loop terminates:
  - If $r == -1$ then $i > j$ so $x \notin S_{l\dots i-1} \wedge x \notin S_{j+1\dots h}$ implies $x \notin S_{l\dots h}$
  - Otherwise **return** m was executed, so $S_m = x$, and so $S_r = x$

- Correctness of recursive algorithms best established using the following particular case of structural induction
- To establish the property $\mathcal{P}(f(x))$ for a recursive function f:
  - Base case: Establish that $\mathcal{P}(f(x))$ holds for all the fixed point(s) (non-recursive case(s)) of f
  - Inductive step: Establish that $\mathcal{P}(f(x))$ holds for all the recursive case(s) of f under the inductive hypothesis that $\mathcal{P}(f(x'))$ is true for all the recursive calls $f(x')$ within f

- Correctness of recursive algorithms best established using the following particular case of structural induction
- To establish the property $\mathcal{P}(f(x))$ for a recursive function f:
  - Base case: Establish that $\mathcal{P}(f(x))$ holds for all the fixed point(s) (non-recursive case(s)) of f
  - Inductive step: Establish that $\mathcal{P}(f(x))$ holds for all the recursive case(s) of f under the inductive hypothesis that $\mathcal{P}(f(x'))$ is true for all the recursive calls $f(x')$ within f
- Technically a structural induction over the recursion tree
  - Also a mathematical induction over the depth of the recursion tree

- Correctness of recursive algorithms best established using the following particular case of structural induction
- To establish the property $\mathcal{P}(f(x))$ for a recursive function f:
  - Base case: Establish that $\mathcal{P}(f(x))$ holds for all the fixed point(s) (non-recursive case(s)) of f
  - Inductive step: Establish that $\mathcal{P}(f(x))$ holds for all the recursive case(s) of f under the inductive hypothesis that $\mathcal{P}(f(x'))$ is true for all the recursive calls $f(x')$ within f
- Technically a structural induction over the recursion tree
  - Also a mathematical induction over the depth of the recursion tree
  - Note in passing: Recursion tree of $f(x)$:
    - Nodes labeled with $f(x)$
    - Node $f(x)$ is the parent of $f(x')$ iff $f(x')$ is (recursively) called from within $f(x)$
    - Leafs are nodes with no recursive calls (fixed points)

EXAMPLE OF STRUCTURAL INDUCTION

algorithm BINSEARCH(x, S, l, h):
- **if** $l > h$ **then return** -1
- **else**
  - $m \leftarrow (l+h)/2$
  - **if** $x == S_m$ **then return** m
  - **else if** $x < S_m$ **then**
    - **return** BINSEARCH(x, S, l, $m-1$)
  - **else** **return** BINSEARCH(x, S, $m+1$, h)

- Base case:

- Need to show that for return r: $S_r = x \vee r = -1 \wedge x \notin S_{l\ldots h}$

Example of structural induction

```
algorithm BinSearch(x, S, l, h):
    if l > h then return -1
    else
        m ← (l + h)/2
        if x == S_m then return m
        else if x < S_m then
            return BinSearch(x, S, l, m − 1)
        else return BinSearch(x, S, m + 1, h)
```

- Need to show that for return r: $S_r = x \vee r = -1 \wedge x \notin S_{l \ldots h}$

- Base case: $l > h$, so the range $S_{l \ldots h}$ is empty, and so $x \notin S_{l \ldots h}$; it is also the case that $r = -1$, as desired
- Inductive hypothesis:

```
algorithm BinSearch(x, S, l, h):
    if l > h then return -1
    else
        m ← (l + h)/2
        if x == S_m then return m
        else if x < S_m then
            return BinSearch(x, S, l, m − 1)
        else  return BinSearch(x, S, m + 1, h)
```

- Need to show that for return r: $S_r = x \vee r = -1 \wedge x \notin S_{l \ldots h}$

- Base case: $l > h$, so the range $S_{l \ldots h}$ is empty, and so $x \notin S_{l \ldots h}$; it is also the case that $r = -1$, as desired
- Inductive hypothesis: The property holds for BINSEARCH(x, S, l, $m - 1$) and BINSEARCH(x, S, $m + 1$, h)
- Inductive step:

```
algorithm BinSearch(x, S, l, h):
    if l > h then return -1
    else
        m ← (l + h)/2
        if x == S_m then return m
        else if x < S_m then
            return BinSearch(x, S, l, m − 1)
        else  return BinSearch(x, S, m + 1, h)
```

- Need to show that for return r: $S_r = x \vee r = -1 \wedge x \notin S_{l\ldots h}$

- Base case: $l > h$, so the range $S_{l\ldots h}$ is empty, and so $x \notin S_{l\ldots h}$; it is also the case that $r = -1$, as desired
- Inductive hypothesis: The property holds for BinSearch(x, S, l, $m-1$) and BinSearch(x, S, $m+1$, h)
- Inductive step:
  - If $S_m = x$ then the appropriate value (m) is returned
  - If $x < S_m$ then $x \notin S_{m\ldots h}$ (see earlier) and so x can only be in $S_{l\ldots m-1}$
    The call BinSearch(x, S, l, $m-1$) will then return the correct r by induction hypothesis
  - If $x > S_m$ then $x \notin S_{l\ldots m}$ (again see earlier) and so x can only be in $S_{m+1\ldots h}$
    The call BinSearch(x, S, $m+1$, h) will then return the correct r by induction hypothesis

ANOTHER EXAMPLE OF STRUCTURAL INDUCTION

```
algorithm MergeSort(S, l, h):
    if l > h then
        m ← (l + h)/2
        MergeSort(S, l, m)
        MergeSort(S, m+1, h)
        Merge(S l, m, h)
```

- Need to show that when MERGESORT(S, l, h) returns the sequence $S_{l\ldots h}$ is sorted
  - Additional assumption: If the sequences $S_{l\ldots m}$ and $S_{m+1\ldots h}$ are sorted before the call MERGE(S l, m, h), then the sequence $S_{l\ldots h}$ is sorted after that call
- Base case: $l \geq h$ means that $S_{l\ldots h}$ holds at most one value so it is already sorted
- Inductive step:
  - Before the call to MERGE the sequences $S_{l\ldots m}$ and $S_{m+1\ldots h}$ are sorted by induction hypothesis
  - Therefore MERGE will return a sorted sequence $s_{l\ldots h}$